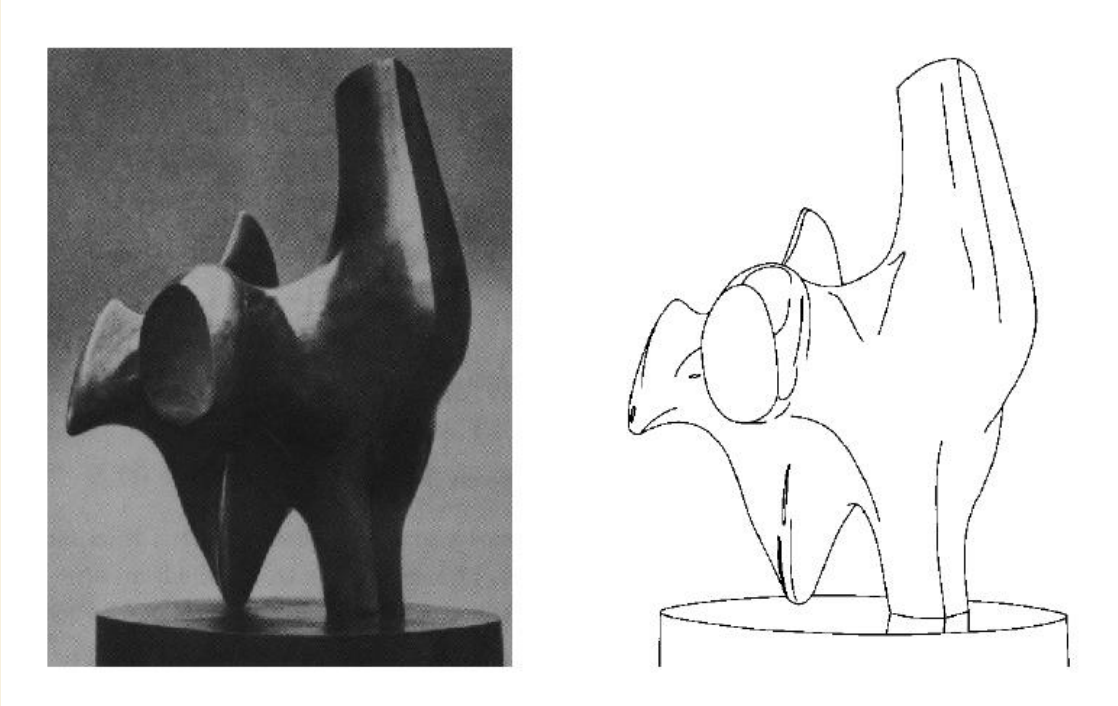
ЭЛЕМЕНТЫ ТЕХНИЧЕСКОГО ЗРЕНИЯ

ЛИНИИ И ОБЛАСТИ НА ИЗОБРАЖЕНИИ

ВЫДЕЛЕНИЕ КРАЕВ



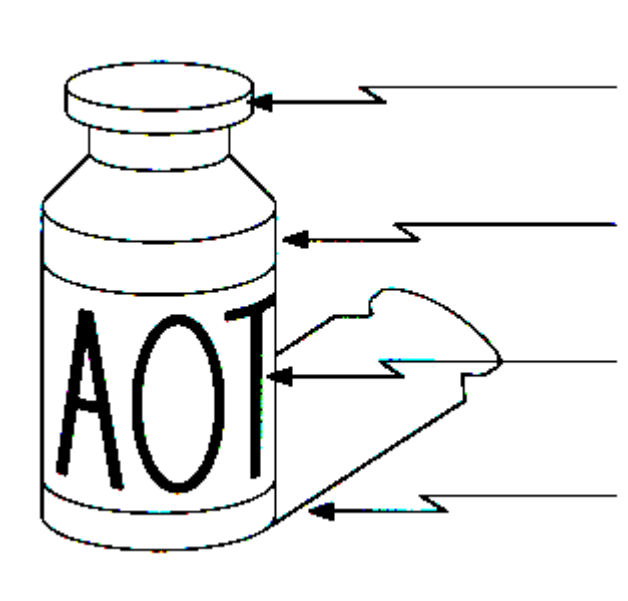
Цель – преобразовать изображение в набор кривых для:

- выделения существенных характеристик
- сокращения объема информации для анализа

ВЫДЕЛЕНИЕ КРАЕВ

ОТКУДА БЕРУТСЯ КРАЯ?

- Край – резкий переход яркости.
- Различные причины возникновения:



Резкое изменение нормали поверхности

Резкое изменение глубины сцены

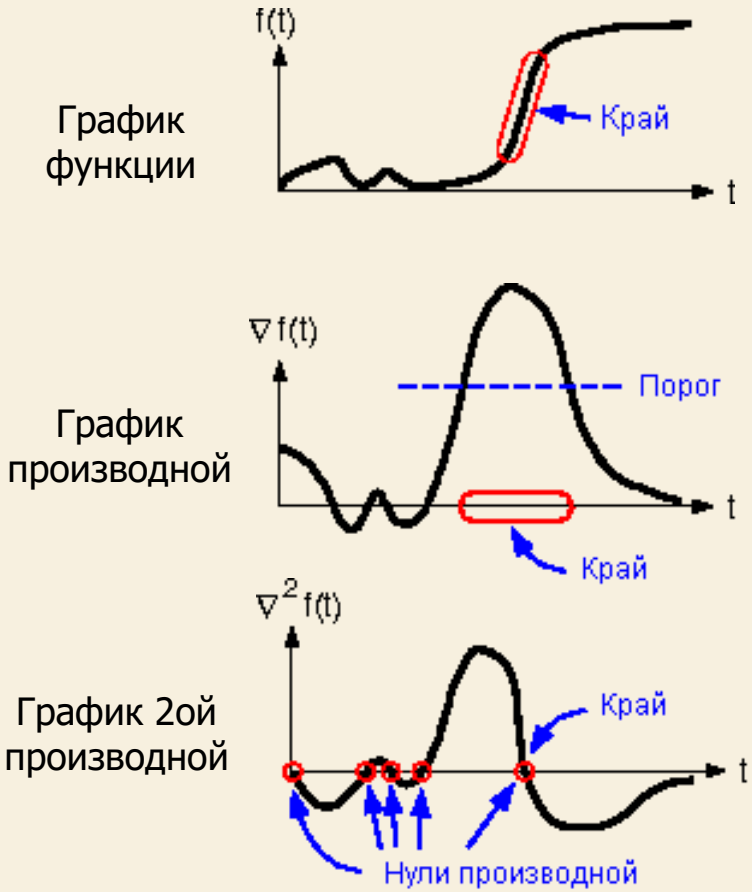
Резкое изменение цвета поверхности

Резкое изменение освещенности

КАК НАЙТИ РЕЗКОЕ ИЗМЕНЕНИЕ ЯРКОСТИ?

Нас интересуют области резкого изменения яркости – нахождение таких областей можно организовать на основе анализа первой и второй производной изображения.

Рассмотрим одномерный случай…



КАК НАЙТИ РЕЗКОЕ ИЗМЕНЕНИЕ ЯРКОСТИ?

Известно, что наибольшее изменение функции происходит в направлении ее градиента. Величина изменения измеряется абсолютной величиной градиента.

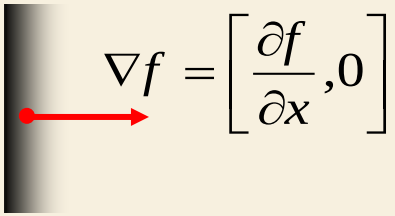
$$\nabla I(x,y) = \left(\frac{\partial I}{\partial x}(x,y), \frac{\partial I}{\partial y}(x,y) \right);$$

$$\left| \nabla I(x,y) \right| = \sqrt{\left(\frac{\partial I}{\partial x}(x,y) \right)^2 + \left(\frac{\partial I}{\partial y}(x,y) \right)^2}$$

ГРАДИЕНТ ЯРКОСТИ ИЗОБРАЖЕНИЯ

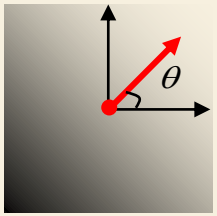
- Известно, что наибольшее изменение функции происходит в направлении ее градиента
  - Приведем примеры…

$$\theta = \arctan\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial y}\right)$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$

$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

ГРАДИЕНТ ЯРКОСТИ ИЗОБРАЖЕНИЯ

- Направление градиента задается:

$$\theta = \arctan\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial y}\right)$$

  «Направление края» задается перпендикулярным градиенту
- «*Сила края*» задается абсолютной величиной градиента:

$$|\nabla I(x,y) = \sqrt{\left(\frac{\partial I}{\partial x}(x,y)\right)^2 + \left(\frac{\partial I}{\partial y}(x,y)\right)^2}$$

- Иногда используется приближенное вычисление градиента:

$$|\nabla I(x,y)| \approx \left|\frac{\partial I}{\partial x}(x,y)\right| + \left|\frac{\partial I}{\partial y}(x,y)\right|$$

ВЫЧИСЛЕНИЕ ГРАДИЕНТА ЯРКОСТИ ИЗОБРАЖЕНИЯ

Семейство методов основано на приближенном вычисление градиента, анализе его направления и абсолютной величины.
Свертка по функциям:

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Робертса

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Превитт

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Собеля

Математический смысл – приближенное вычисление производных по направлению

КАРТА СИЛЫ КРАЕВ

Примеры:

Робертса

Превитт

Собеля

РЕАЛИЗАЦИЯ В OPENCV

```cpp
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
    Mat image,src, src_gray, grad;
    const String window_name = "Gradient";
    int ksize = 3;
    int scale = 1;
    int delta = 0;
    int ddepth = CV_16S;

    String imageName =("Lenna.png");
    image = imread(imageName, IMREAD_COLOR );
    if( image.empty() )
    {
        return EXIT_FAILURE;
    }

    GaussianBlur(image, src, Size(3, 3), 0, 0, BORDER_DEFAULT);
    cvtColor(src, src_gray, COLOR_BGR2GRAY);

    Mat grad_x, grad_y;
    Mat abs_grad_x, abs_grad_y;

    Sobel(src_gray, grad_x, ddepth, 1,0,ksize, scale, delta, BORDER_DEFAULT);
    Sobel(src_gray, grad_y, ddepth, 0,1,ksize, scale, delta, BORDER_DEFAULT);

    // Преобразование в CV_8U
    convertScaleAbs(grad_x, abs_grad_x);
    convertScaleAbs(grad_y, abs_grad_y);
    //  Вычисление градиента - складываем с весами 0.5 и добавкой 0
    addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, grad);

    imshow(window_name, grad);
    char key = (char)waitKey(0);
    return EXIT_SUCCESS;
}
```

SOBEL И SCHARR

- Синтаксис

Sobel(InputArray src, OutputArray dst, int ddepth, int dx, int dy,

int ksize = 3, double scale = 1, double delta = 0, int borderType = BORDER_DEFAULT)

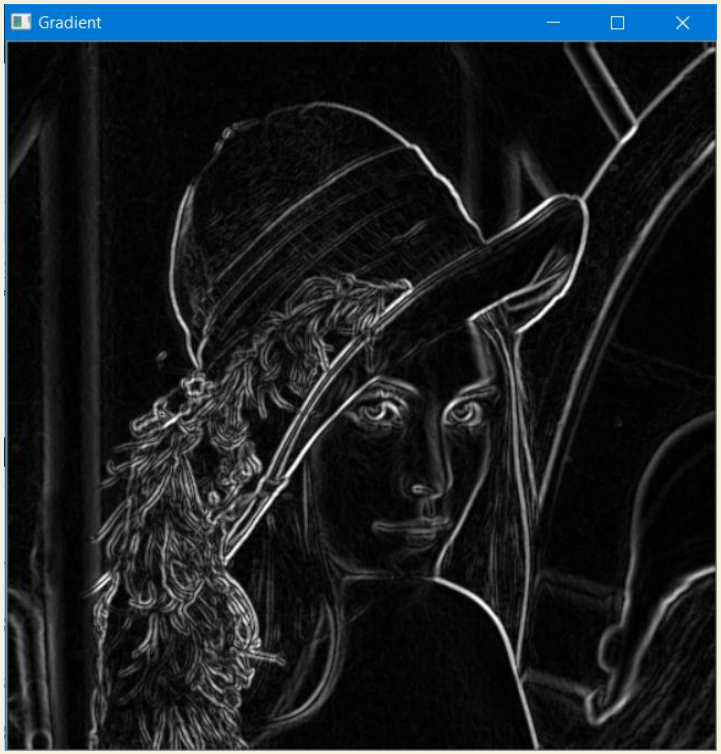
- Аналогичный оператор Sharr с фиксированым ядром (ksize) 3x3

Scharr(InputArray src, OutputArray dst, int ddepth, int dx, int dy,

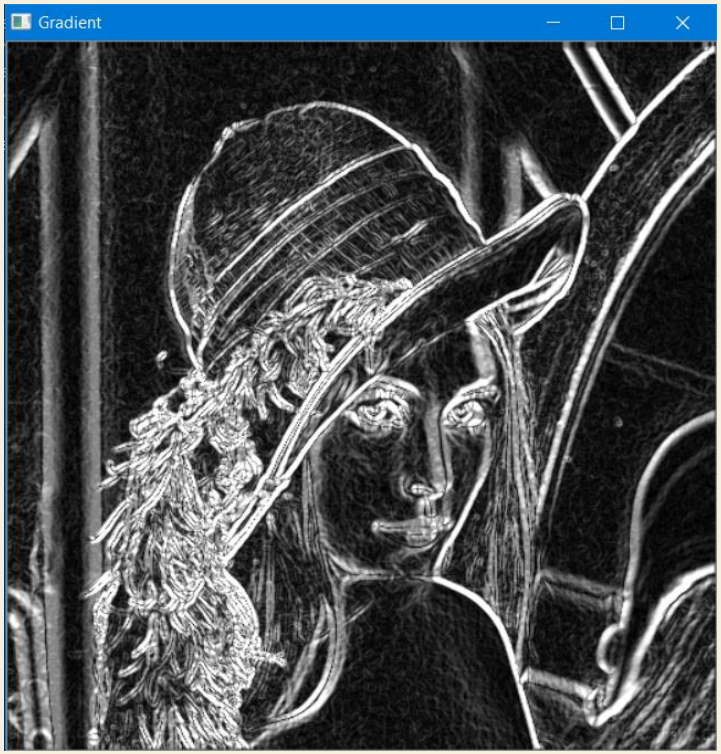
double scale = 1, double delta = 0, int borderType = BORDER_DEFAULT)

РЕЗУЛЬТАТ

SOBEL



SCHARR



ВЫДЕЛЕНИЕ КРАЕВ

- Вычисление градиента – это еще не всё…

Исходное изображение

Карта силы краев

- Чего не хватает?
  - Точности – края «толстые» и размытые
  - Информации о связности

ВЫДЕЛЕНИЕ КРАЕВ

- Нужно:
  - Убрать слабые края и шум
  - Сделать края тонкими
  - Объединить пиксели краев в связные кривые

АЛГОРИТМ CANNY

- Давно придуман, однако до сих пор широко используется
- Шаги:
  - Убрать шум и лишние детали из изображения
  - Рассчитать градиент изображения
  - Сделать края тонкими (edge thinning)
  - Связать края в контура (edge linking)

НАЧАЛО…

- Размыть изображение с помощью фильтра Гаусса
  - Убрать шум, лишние детали текстуры
- Рассчитать градиент изображения
  - Одним из операторов – например, Собеля

Исходное изображение

Размытое изображение

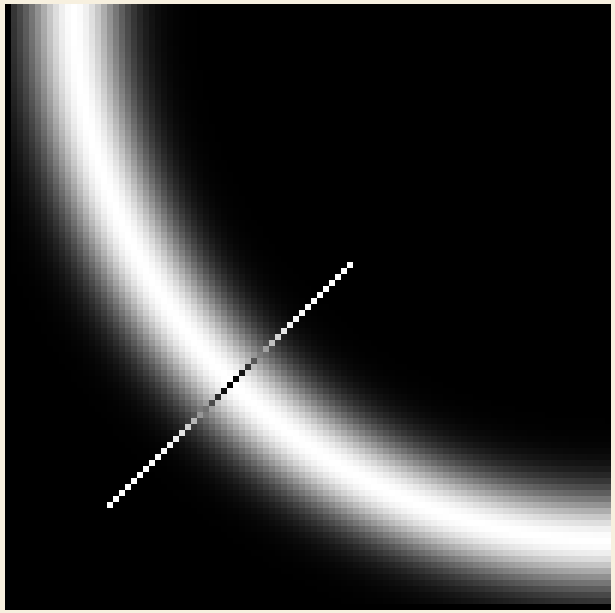
Карта силы краев

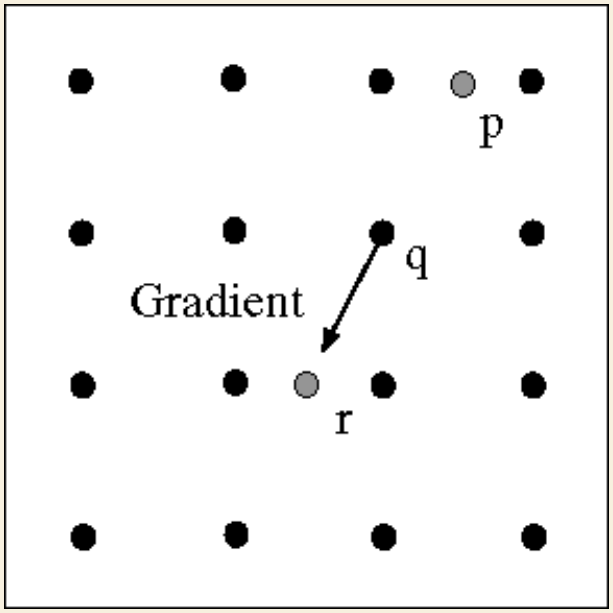
...СЕРЕДИНА...

Все пиксели где сила краев < *T* убрать из рассмотрения

…СЕРЕДИНА…

- Поиск локальных максимумов





- Проверяя – является ли пиксель локальным максимумом вдоль направления градиента
  - Приходится интерполировать «нецелые» пиксели p и r

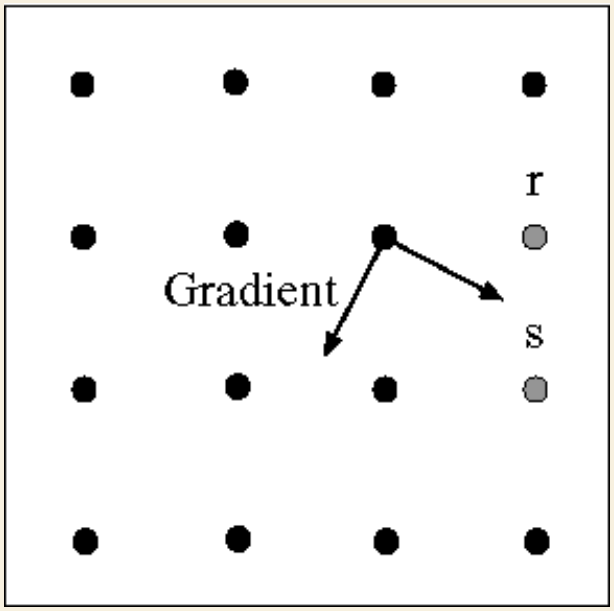
… ФИНАЛ

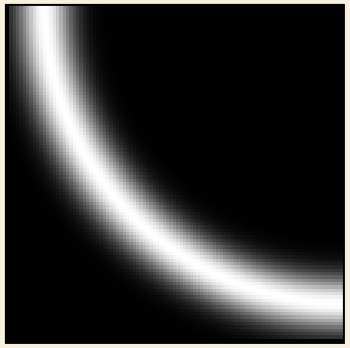
1. Выбираем еще не обработанную точку локального максимума p в которой сила края $\left|\nabla I(p)\right| > T_1$
2. Прослеживание края выбранного локального максимума p:
   1. Предсказание следующей точки края q;
   2. Проверка – $\left|\nabla I(q)\right| > T_2$? $(T_1 > T_2)$
   3. Если да – $p = q$, переход на начало шага 2;
   4. Если нет - переход на шаг 1;

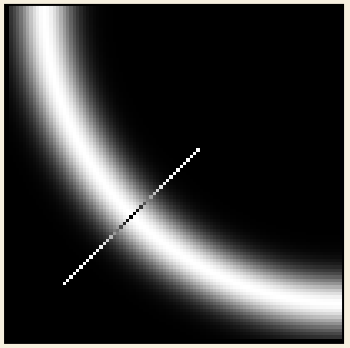
ПОЯСНЕНИЯ

Как предсказать следующую точку края?

- От текущей точки шаг в сторону перпендикулярную градиенту
- В данном случае – в точку r или s



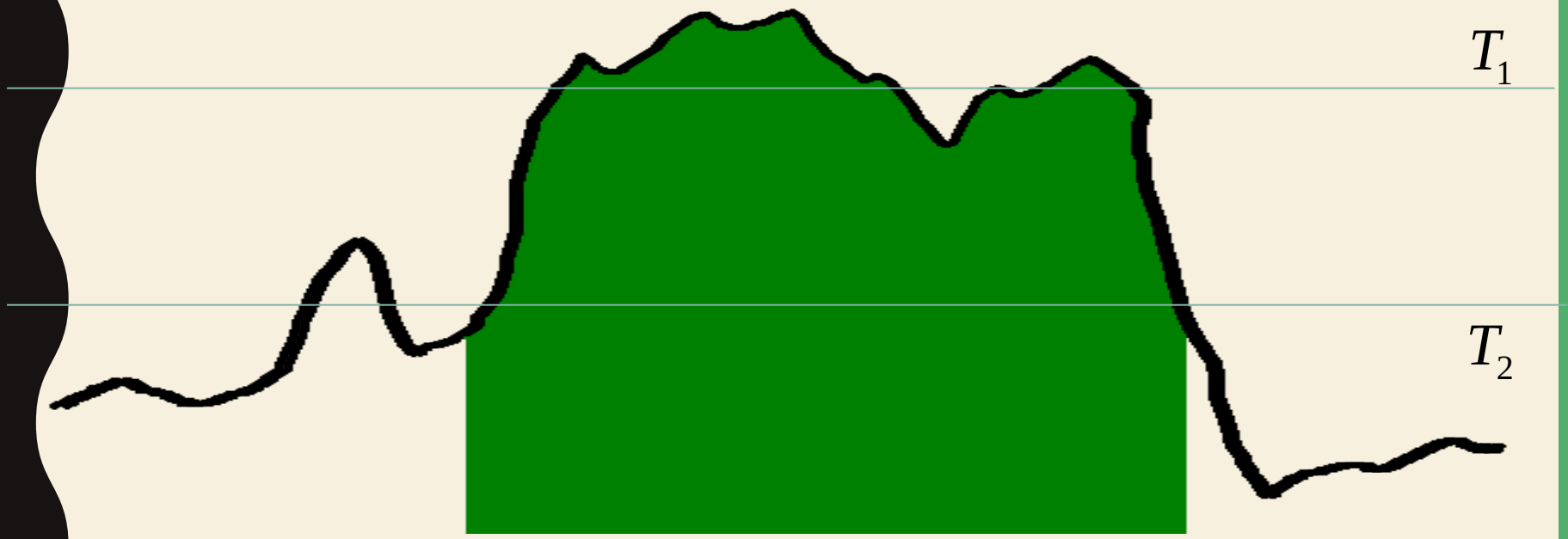




ПОЯСНЕНИЯ

Для чего используются два порога $T_1 > T_2$?

- Чтобы не потерять хвост
- Чтобы уменьшить влияние шума для инициализации кривой используем больший порог
- используем меньший порог при прослеживании



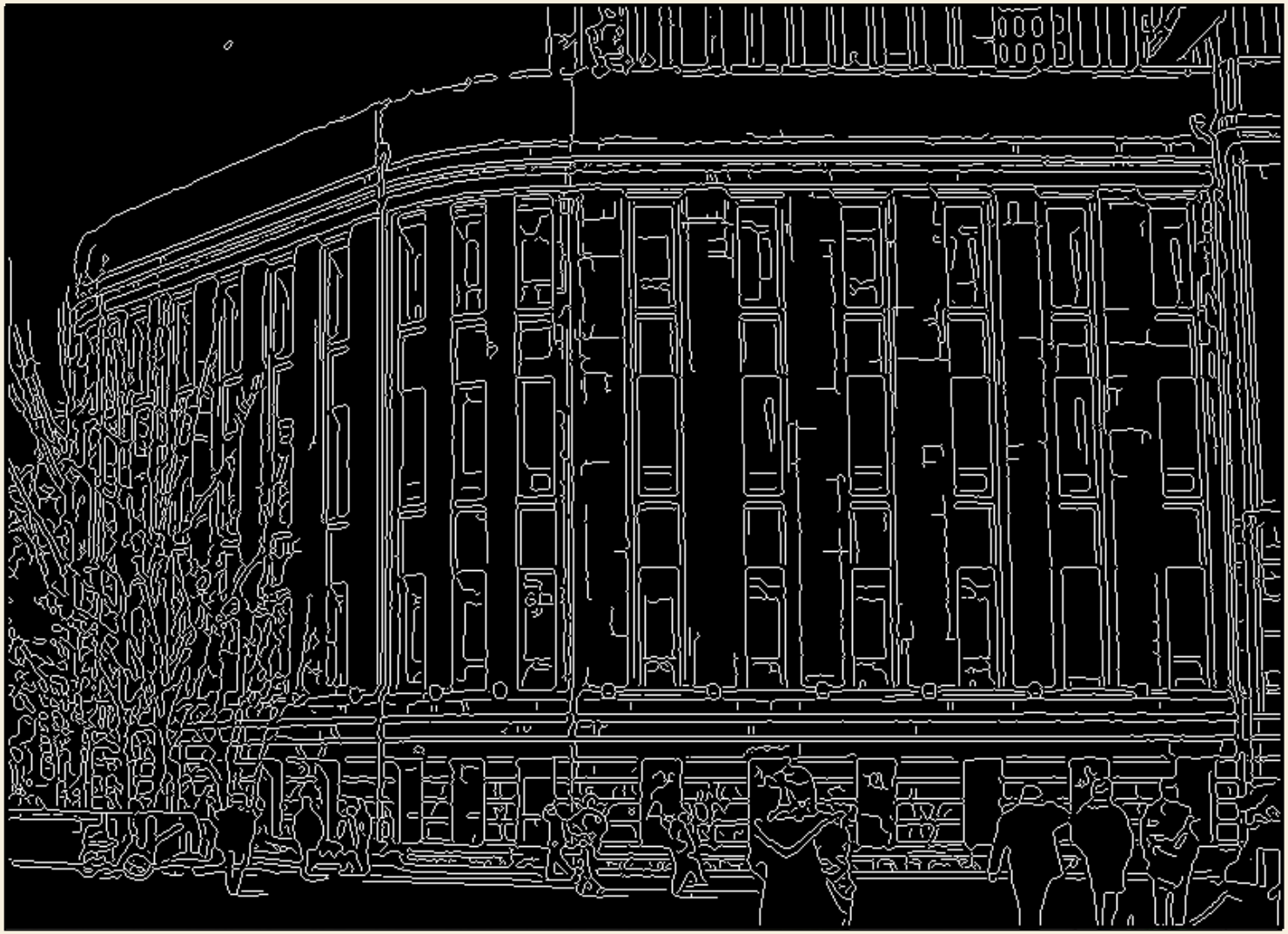
АЛГОРИТМ CANNY

- Размыть изображение фильтром Гаусса с некоторым σ
  - Убрать шум, лишние детали текстуры
- Рассчитать градиент изображения
  - Одним из операторов – например, Собеля
- Все пиксели где сила краев $< T$ убрать из рассмотрения
- Поиск локальных максимумов
- Прослеживание краев из точек локальных максимумов

CANNY – ИСХОДНЫЙ

CANNY - КРАЯ

CANNY - РЕЗУЛЬТАТ



ВЛИЯНИЕ σ (ПАРАМЕТР ФИЛЬТРА ГАУССА)

исходное

Canny с $\sigma = 1$

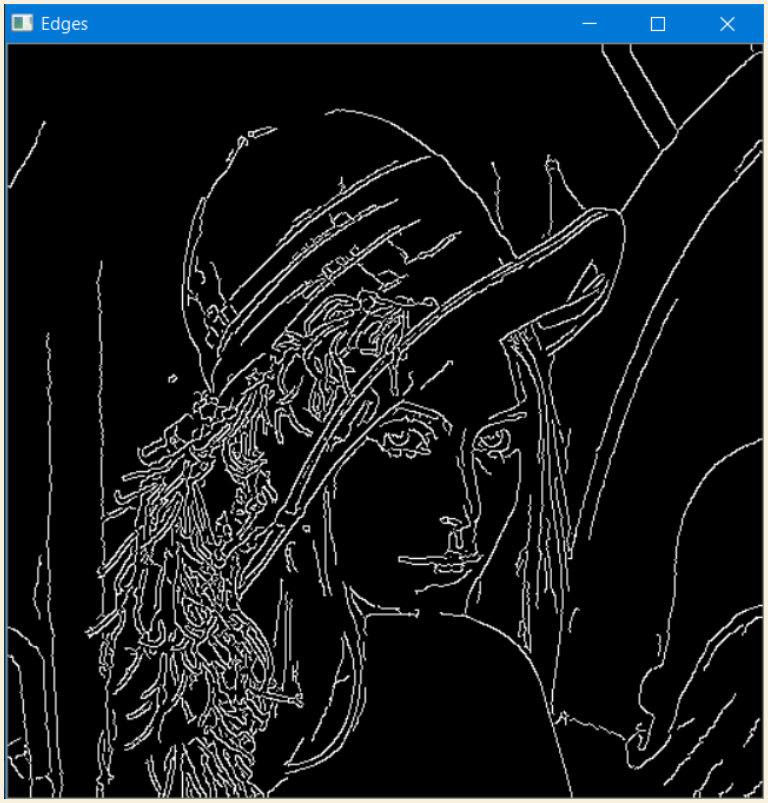
Canny с $\sigma = 2$

РЕАЛИЗАЦИЯ С ПОМОЩЬЮ OPENCV

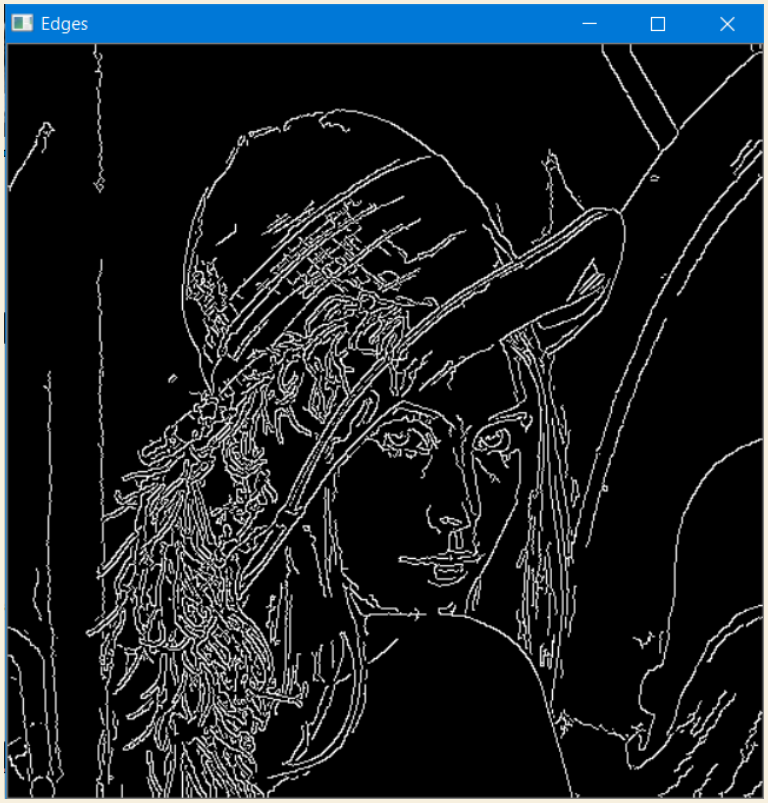
```
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"

#include <iostream>

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
    Mat image,src, src_gray, grad;
    const String window_name = "Edges";
    int ksize = 3;
    int scale = 1;
    int delta = 0;
    int ddepth = CV_16S;

    String imageName =("Lenna.png");
    image = imread(imageName, IMREAD_COLOR );
    if( image.empty() )
    {
        return EXIT_FAILURE;
    }

    GaussianBlur(image, src, Size(3, 3), 0, 0, BORDER_DEFAULT);
    cvtColor(src, src_gray, COLOR_BGR2GRAY);

    Canny(src_gray, grad, 30, 90);

    imshow(window_name, grad);
    char key = (char)waitKey(0);

    return EXIT_SUCCESS;
}
```

РЕЗУЛЬТАТ С РАЗЛИЧНЫМ СГЛАЖИВАНИЕМ

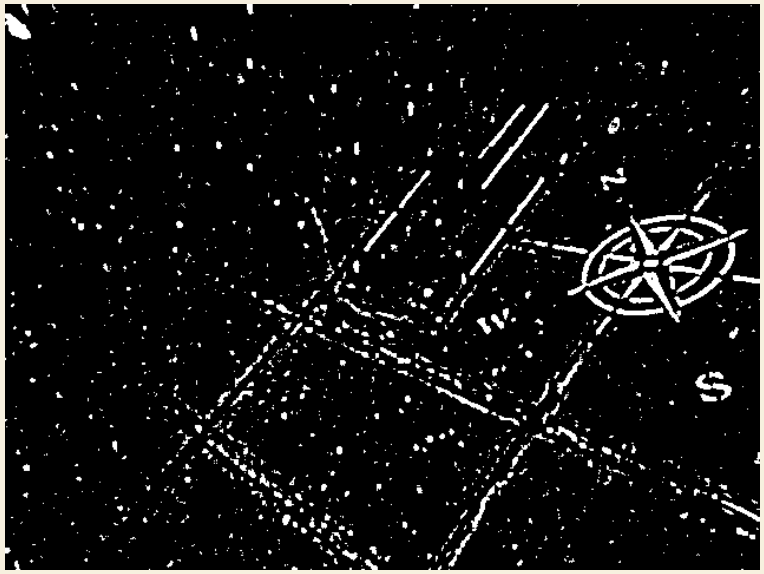
5X5

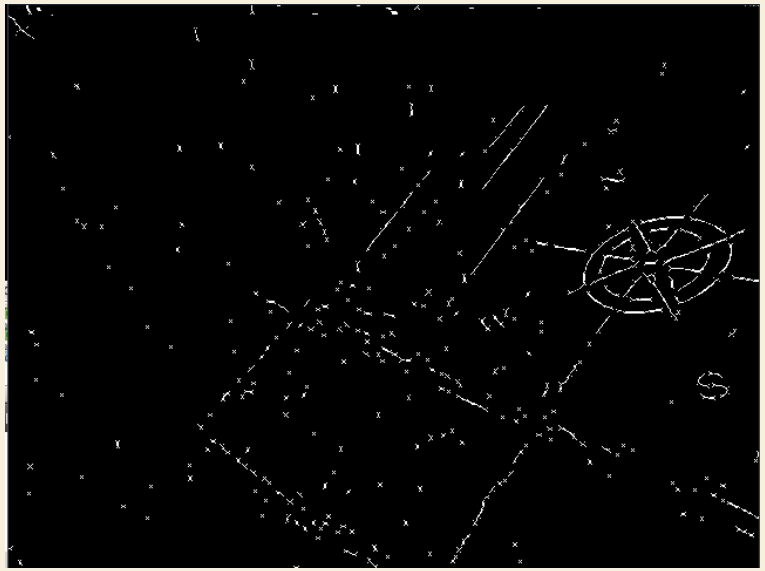


3X3



МОРФОЛОГИЧЕСКИЕ ОПЕРАЦИИ



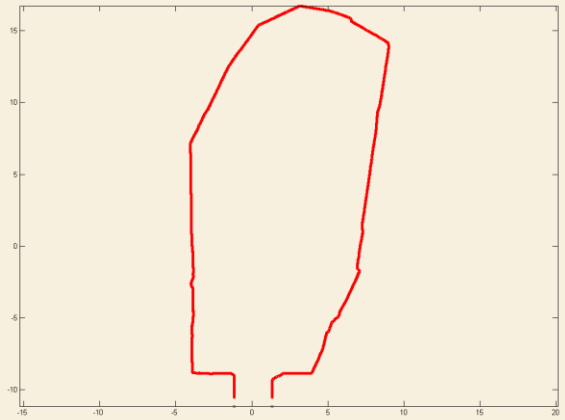


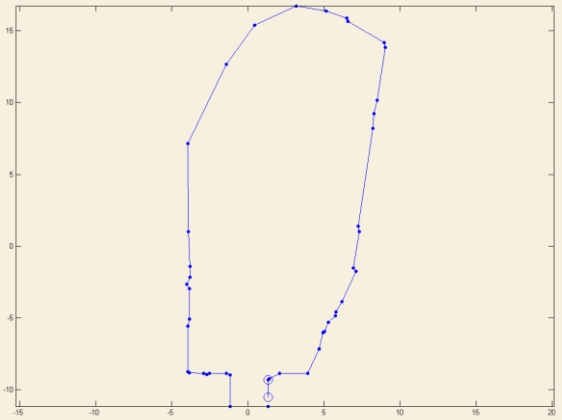
РАБОТА С КОНТУРАМИ

- Полигональная аппроксимация
- Цепные коды
- Дескрипторы контуров

ПОЛИГОНАЛЬНАЯ АППРОКСИМАЦИЯ

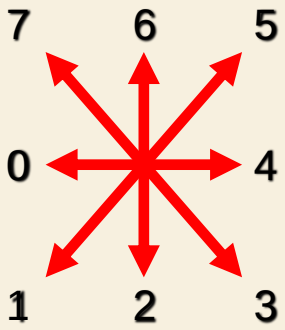
- Постановка:
  - Аппроксимация точечной кривой ломаной линией
- Цель:
  - Сжатие информации
  - Борьба с дискретностью и шумом
  - Облегчение дальнейшего анализа

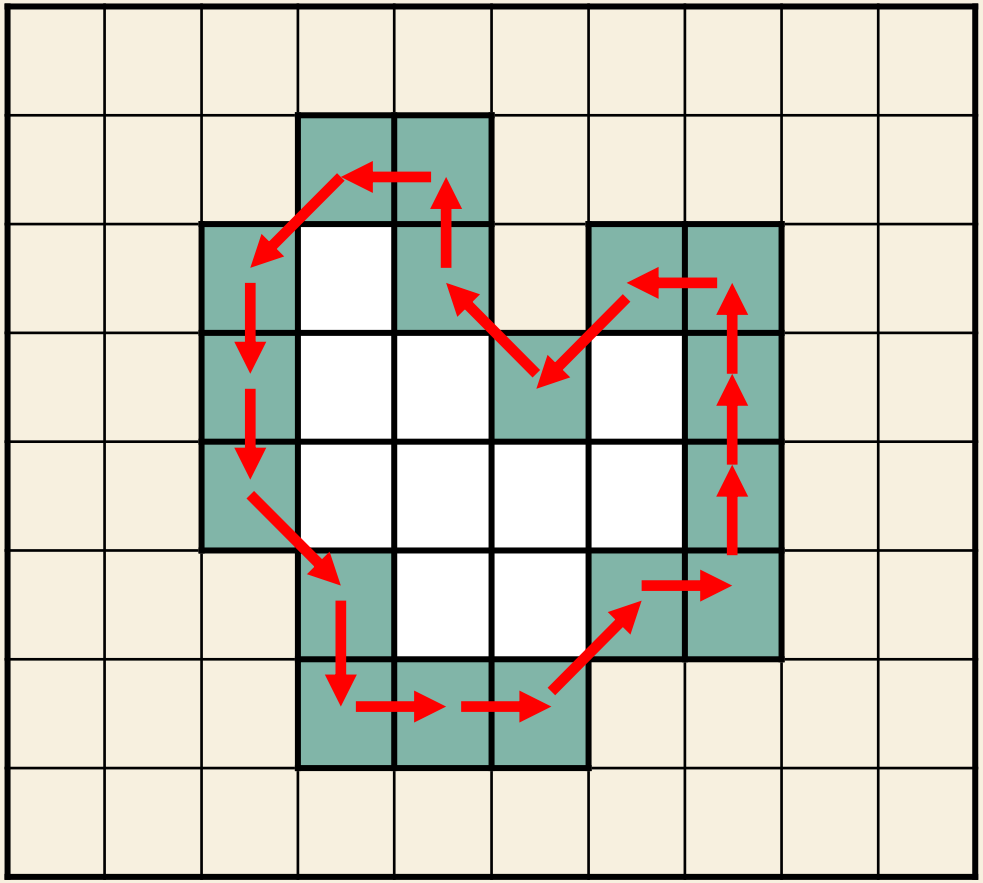




ЦЕПНОЙ КОД 8-СВЯЗНЫЕ КОНТУРА

- Кодирование контура как последовательности перемещений

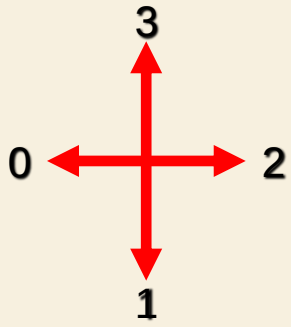


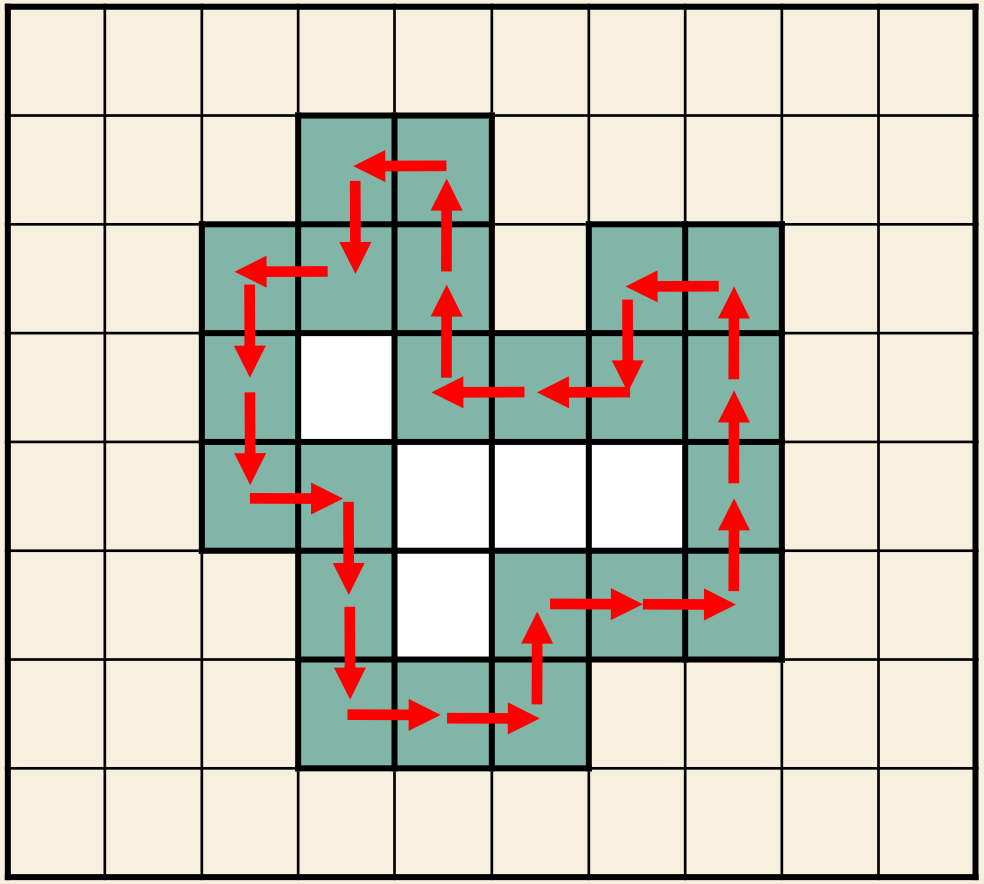


Код: 12232445466601760

ЦЕПНОЙ КОД 4-СВЯЗНЫЕ КОНТУРА

- Кодирование контура как последовательности перемещений





Код: 11223223330100330101112

ЦЕПНОЙ КОД - СВОЙСТВА

- Свойства
  - Цепной код – представление контура, независимое к его перемещению
  - При замене в 8-ми связном кода любого n но (n mod 8) + 1 контур будет **повернут** по часовой стрелке на 45 градусов
  - Некоторые особенности контуров, такие как уголки, например могут быть сразу рассчитаны по анализу цепных кодов

- Сложности
  - В цепном коде важна начальная точка – при ее изменении меняется и код
  - Небольшие вариации границы (шум) серьезно меняют код. Сравнение двух шумных контуров по цепному коды – сложно
  - Цепной код не инвариантен к повороту

РАЗНОСТНЫЙ КОД

- Это «производная» цепного кода
  - Формула
    - $y_i = (x_{i+1} - x_i) \bmod 8$ – восьмисвязный
    - $y_i = (x_{i+1} - x_i) \bmod 4$ – четырехсвязный

ВЫДЕЛЕНИЕ ОБЛАСТЕЙ

- Может производиться различными способами
  - Выделение с помощью цветовых пространств (гистограмма)
  - Частный случай – результат бинаризации
  - Внутренность контура
  - Сегментация

ПОСТРОЕНИЕ ВЫПУКЛЫХ ОБОЛОЧЕК

Пусть нам удалось построить контур (как обсуждали раньше),

Мы можем построить выпуклый многоугольник, который этот контур охватывает

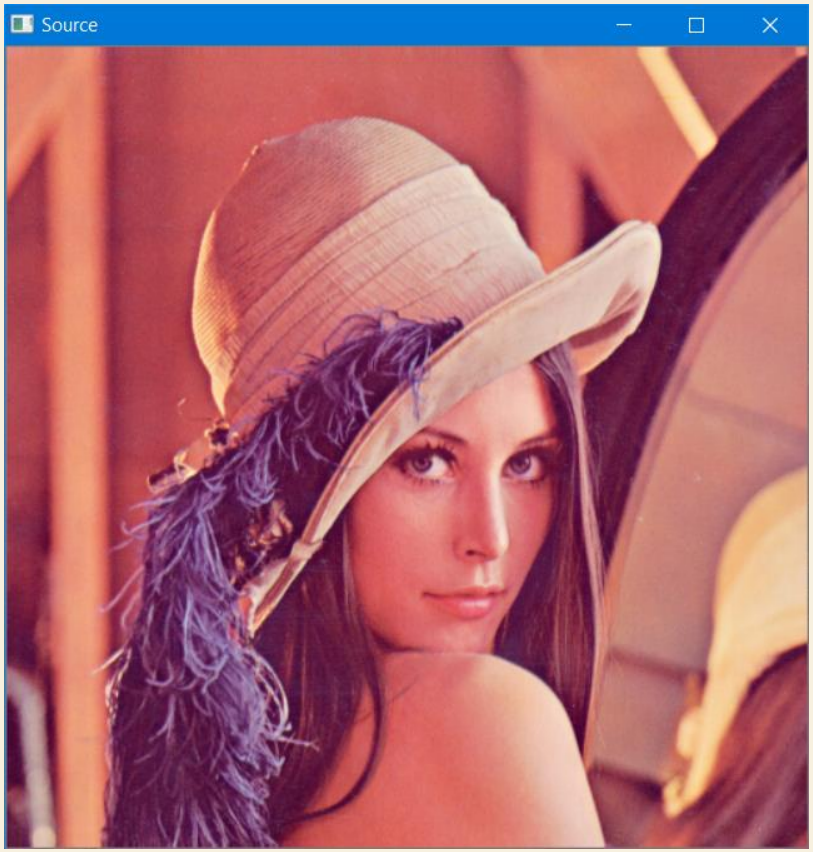
Если контур не замкнут мы замкнём разрывы

```cpp
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
    RNG rng(12345);
    Mat image,src, src_gray, grad;
    int ksize = 3;
    int scale = 1;
    int delta = 0;
    int ddepth = CV_16S;

    String imageName =("Lenna.png");
    image = imread(imageName, IMREAD_COLOR ); // Load an image
    if( image.empty() )
    {
        printf("Error opening image: %s\n", imageName.c_str());
        return EXIT_FAILURE;
    }

    GaussianBlur(image, src, Size(9, 9), 0, 0, BORDER_DEFAULT);
    cvtColor(src, src_gray, COLOR_BGR2GRAY);
    Canny(src_gray, grad, 30, 90);

    vector<vector<Point> > contours;
    findContours( grad, contours, RETR_TREE, CHAIN_APPROX_SIMPLE );

    vector<vector<Point> >hull( contours.size() );
    for( size_t i = 0; i < contours.size(); i++ )
    {
        convexHull( contours[i], hull[i] );
    }

    Mat drawing = Mat::zeros(grad.size(), CV_8UC3 );
    for( size_t i = 0; i< contours.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 256), rng.uniform(0,256), rng.uniform(0,256) );
        drawContours( drawing, contours, (int)i, color );
        drawContours( drawing, hull, (int)i, color );
    }
    imshow("hull", drawing);
    char key = (char)waitKey(0);
    return EXIT_SUCCESS;
}
```

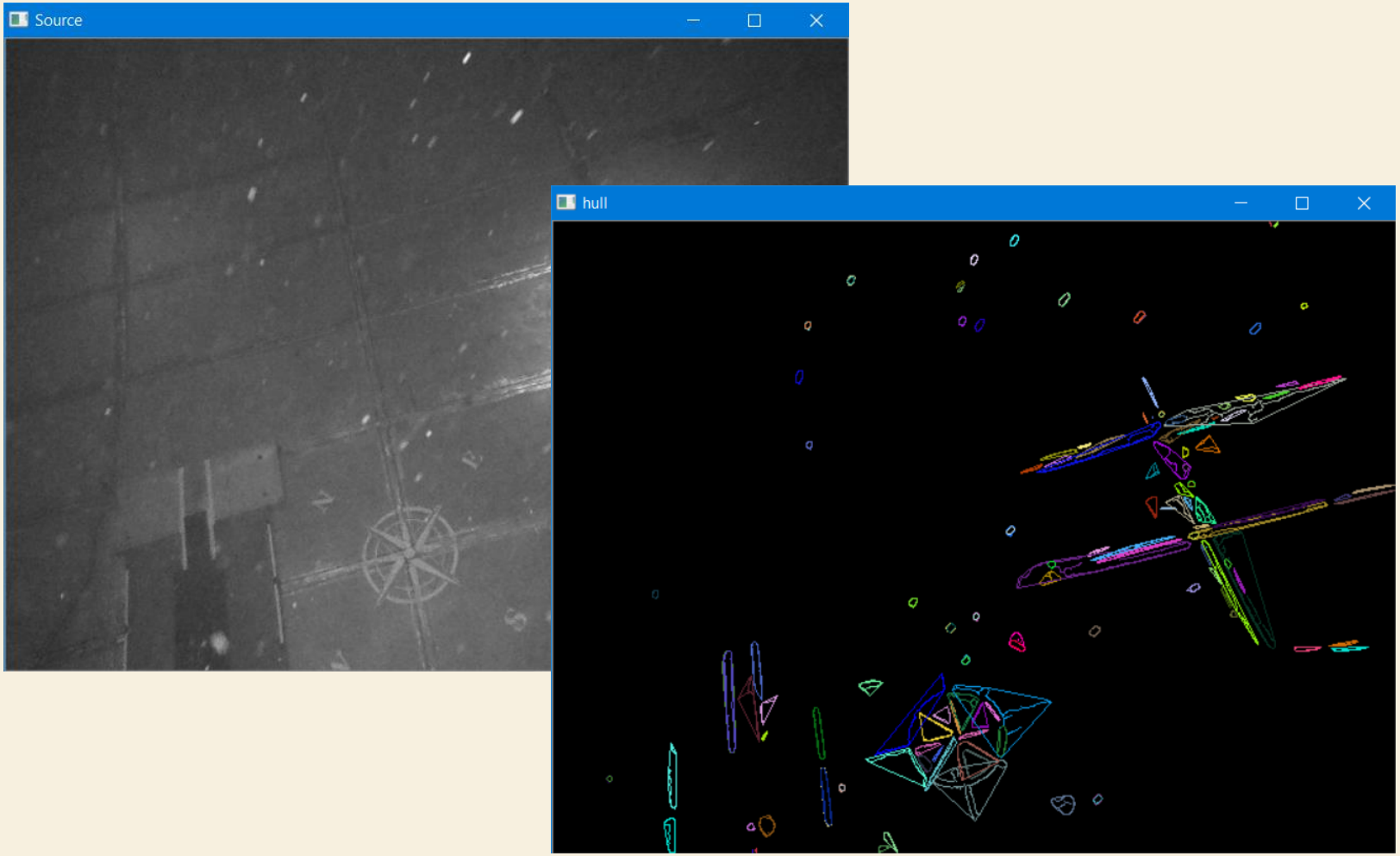
ПОИСК КОНТУРОВ И ОБОЛОЧЕК

- findContours ищет контура с использованием цепного кода
- convexHull строит по ним оболочки

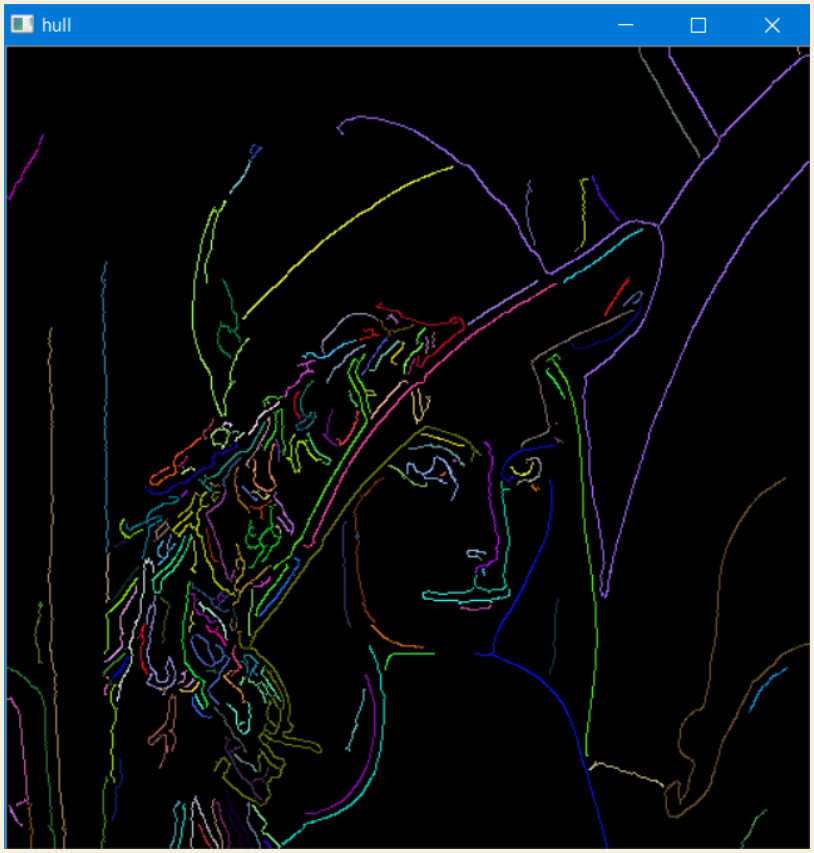
ИЗОБРАЖЕНИЕ И КРАЯ

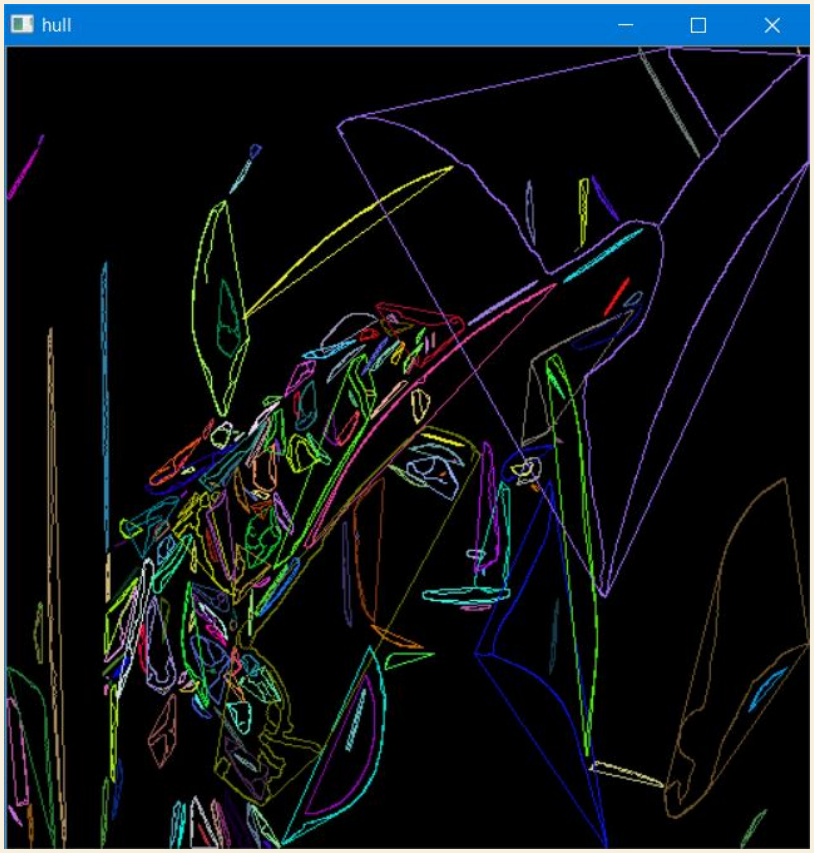


БОЛЕЕ РЕАЛИСТИЧНОЕ ИЗОБРАЖЕНИЕ



КОНТУРЫ И ПОЛИГОНЫ





О ПРОБЛЕММАХ С OPENCV В ВИРТУАЛЬНОЙ МАШИНЕ

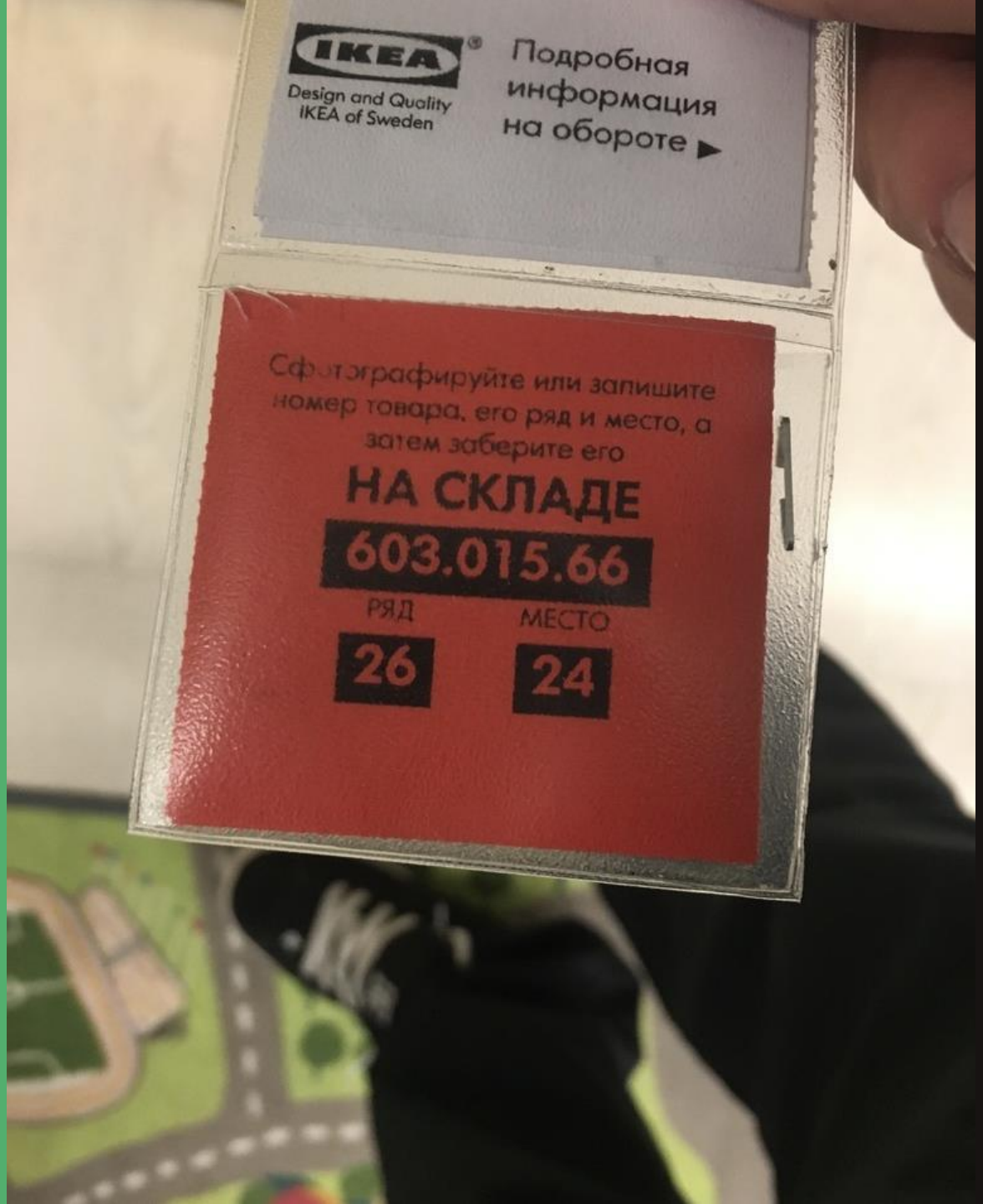
- Возникла проблемма с местом установки бибоиотеки
- Путь изменился:

```
INCLUDEPATH += /usr/local/include
LIBS += -L/usr/local/lib \
        -l:libopencv_core.so.3.4
        и т.д.
```

ПРО РАСПОЗНАВАНИЕ ТЕКСТА

- OpenCV может распознавать текст с использованием библиотеки tesseract (если он правильно собран)

```
#include <opencv2/text.hpp>
Ptr<text::BaseOCR> ocr = text::OCRTesseract::create(nullptr, "rus");
string text;
ocr->run(img, text);
cout << "Recognized text: " << text << endl;
```



ЗАДАНИЕ 5

- Выделите на изображении фрагменты с кодом, рядом и местом
- Опционально: попробуйте распознать текст!